CHAPTER 2

CHAPTER 2 *CDMS Python Application Programming Interface*

2.1 Overview

This chapter describes the CDMS Python application programming interface (API). Python is a popular public−domain, object−oriented language. Its features include support for object−oriented development, a rich set of programming constructs, and an extensible architecture. CDMS itself is implemented in a mixture of C and Python. In this chapter the assumption is made that the reader is familiar with the basic features of the Python language.

Python supports the notion of a **module**, which groups together associated classes and methods. The **import** command makes the module accessible to an application. This chapter documents the **cdms, cdtime,** and **regrid** modules.

The chapter sections correspond to the CDMS classes. Each section contains tables base. If no parent, the datapath is absolute.describing the class internal (non−persistent) attributes, constructors (functions for creating an object), and class methods (functions). A method can return an instance of a CDMS class, or one of the Python types:

Table 2.1 Python types used in CDMS

Type	Description
Array	Numeric or masked multidimensional data array. All elements of the array are of the same type. Defined in the **Numeric** and **MA** modules.
Comptime	Absolute time value, a time with representation (year, month, day, hour, minute, second). Defined in the **cdtime** module. cf. **reltime**
Dictionary	An unordered 2,3collection of objects, indexed by key. All dictionaries in CDMS are indexed by strings, e.g.: **axes['time']**
Float	Floating−point value.
Integer	Integer value.
List	An ordered sequence of objects, which need not be of the same type. New members can be inserted or appended. Lists are denoted with square brackets, e.g., **[1, 2.0, 'x', 'y']**
None	No value returned.
Reltime	Relative time value, a time with representation (value, units since basetime). Defined in the **cdtime** module. cf. **comptime**

Tuple — An ordered sequence of objects, which need not be of the same type. Unlike lists, tuples elements cannot be inserted or appended. Tuples are denoted with parentheses, e.g.,

(1, 2.0, 'x', 'y')

2.2 A first example

The following Python script reads January and July monthly temperature data from an input dataset, averages over time, and writes the results to an output file. The input temperature data is ordered (time, latitude, longitude).

```
1 #!/usr/bin/env python
2 import cdms
3 from cdms import MV
4 jones = cdms.open('/pcmdi/cdms/obs/jones_mo.nc')
5 tasvar = jones['tas']
6 jans = tasvar[0::12]
7 julys = tasvar[6::12]
8 janavg = MV.average(jans)
9 janavg.id = "tas_jan"
10 janavg.long_name = "mean January surface temperature"
11 julyavg = MV.average(julys)
12 julyavg.id = "tas_jul"
13 julyavg.long_name = "mean July surface temperature"
14 out = cdms.open('janjuly.nc','w')
15 out.write(janavg)
16 out.write(julyavg)
17 out.comment = "Average January/July from Jones dataset"
18 jones.close()
19 out.close()
```

Line	**Notes**
2,3	Makes the CDMS and MV modules available. MV defines arithmetic functions.
4	Opens a netCDF file read−only. The result **jones** is a dataset object.
5	Gets the surface air temperature variable. '**tas'** is the name of the variable in the input dataset. This does not actually read the data.
6	Read all January monthly mean data into a variable **jans**. Variables can be sliced like arrays. The slice operator **[0::12]** means take every 12th slice from dimension 0, starting at index 0 and ending at the last index. If the stride **12** were omitted, it would default to 1. Note that the variable is actually 3−dimensional. Since no slice is specified for the second or third dimensions, all values of those2,3 dimensions are retrieved. The slice operation could also have been written **[0::12, : , :]**. Also note that the same script works for multi−file datasets. CDMS opens

	the needed data files, extracts the appropriate slices, and concatenates them into the result array.
7	Reads all July data into a masked array **julys**.
8	Calculate the average January value for each grid zone. Any missing data is handled automatically.
9,10	Set the variable **id** and **long_name** attributes. The id is used as the name of the variable when plotted or written to a file.
14	Create a new netCDF output file named '**janjuly.nc'** to hold the results.
15	Write the January average values to the output file. The variable will have id **"tas_jan"** in the file. **write** is a utility function which creates the variable in the file, then writes data to the variable. A more general method of data output is first to create a variable, then set a slice of the variable. Note that **janavg** and **julavg** have the same latitude and longitude information as **tasvar**. It is carried along with the computations.
17	Set the global attribute '**comment'**.
18	Close the output file.

2.3 cdms module

The **cdms** module is the Python interface to CDMS. The objects and methods in this chapter are made accessible with the command:

import cdms

The functions described in this section are not associated with a class. Rather, they are called as module functions, e.g.,

file = cdms.open('sample.nc')

Table 2.2 cdms module functions

Type **Definition**

Variable **asVariable(s)**

Transform s into a transient variable.

s is a masked array, Numeric array, or Variable. If s is already a transient variable, s is returned.

See also: **isVariable**.

Axis **createAxis(data, bounds=None)**

Create a one−dimensional coordinate Axis, which is not associated with a file or dataset. This is useful for creating a grid which is not contained in a file or dataset.

data is a one−dimensional, monotonic Numeric array.

bounds is an array of shape (len(data),2), such that for all i, data[i] is in the range [bounds[i,0],bounds[i,1]]. If bounds is not specified, the default boundaries are generated at the midpoints between the consecutive data values, provided that the autobounds mode is 'on' (the default). See **setAutoBounds**.

Also see: **CdmsFile.createAxis**

Axis **createEqualAreaAxis(nlat)**

Create an equal−area latitude axis. The latitude values range from north to south, and for all axis values x[i], sin(x[i])sin(x[i+1]) is constant.

nlat is the axis length.

The axis is not associated with a file or dataset.

Axis **createGaussianAxis(nlat)**

Create a Gaussian latitude axis. Axis values range from north to south.

nlat is the axis length.

The axis is not associated with a file or dataset.

RectGrid **createGaussianGrid(nlats, xorigin=0.0, order="yx")**
Create a Gaussian grid, with shape (nlats, 2*nlats).

nlats is the number of latitudes.
xorigin is the origin of the longitude axis.
order is either "yx" (lat−lon, default) or "xy" (lon−lat)

createGenericGrid(latArray, lonArray, latBounds=None, lonBounds=None, order="yx", mask=None)

RectGrid

Create a generic grid, that is, a grid which is not typed as Gaussian, uniform, or equal−area. The grid is not associated with a file or dataset.

latArray is a NumPy array of latitude values.
lonArray is a NumPy array of longitude values
latBounds is a NumPy array having shape (len(latArray),2), of latitude boundaries.
lonBounds is a NumPy array having shape (len(lonArray),2), of longitude boundaries.
order is a string specifying the order of the axes, either "yx" for (latitude, longitude), or "xy" for the reverse.
mask (optional) is an integer−valued NumPy mask array, having the same shape and ordering as the grid.

createGlobalMeanGrid(grid)

RectGrid

Generate a grid for calculating the global mean via a regridding operation. The return grid is a single zone covering the range of the input grid.

grid is a RectGrid.

createRectGrid(lat, lon, order, type="generic", mask=None)

RectGrid

Create a rectilinear grid, not associated with a file or dataset.
This might be used as the target grid for a regridding operation.

lat is a latitude axis, created by cdms.createAxis.
lon is a longitude axis, created by cdms.createAxis.
order is a string with value "yx" (the first grid dimension is latitude) or "xy" (the first grid dimension is longitude).
type is one of 'gaussian','uniform','equalarea',or 'generic'.
If specified, *mask* is a two−dimensional, logical Numeric array (all values are zero or one) with the same shape as the grid.

RectGrid **createUniformGrid(startLat, nlat, deltaLat, start−Lon, nlon, deltaLon, order="yx", mask=None)**
Create a uniform rectilinear grid. The grid is not associated
with a file or dataset. The grid boundaries are at the midpoints
of the axis values.

startLat is the starting latitude value.
nlat is the number of latitudes. If nlat is 1, the grid latitude boundaries will be startLat +/− deltaLat/2.
deltaLat is the increment between latitudes.
startLon is the starting longitude value.
nlon is the number of longitudes. If nlon is 1, the grid longitude boundaries will be startLon +/−deltaLon/2.
deltaLon is the increment between longitudes.
order is a string with value "y"x (the first grid dimension is latitude) or "xy" (the first grid dimension is longitude). .
If specified, *mask* is a two−dimensional, logical Numeric array
(all values are zero or one) with the same shape as the grid.

Axis **createUniformLatitudeAxis(startLat, nlat, deltaLat)**

Create a uniform latitude axis. The axis boundaries are at the midpoints of the axis values. The axis is designated as a circular latitude axis.

startLat is the starting latitude value.

nlat is the number of latitudes.

deltaLat is the increment between latitudes.

RectGrid **createZonalGrid(grid)**

Create a zonal grid. The output grid has the same latitude as the input grid, and a single longitude. This may be used to calculate zonal averages via a regridding operation.
grid is a RectGrid.

Axis **createUniformLongitudeAxis(startLon, nlon, delta−Lon)**
Create a uniform longitude axis. The axis boundaries are at the midpoints of the axis values. The axis is designated as a circular longitude axis.

startLon is the starting longitude value.

nlon is the number of longitudes.

deltaLon is the increment between longitudes.

Variable **createVariable(array, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, grid=None, axes=None, attributes=None, id=None)**
This function is documented in Table 2.34 on page 90.

Integer **getAutoBounds()**

Get the current autobounds mode. Returns 0, 1, or 2. See **setAutoBounds**.

Integer **isVariable(s)**

Return 1 if s is a variable, 0 otherwise. See also: **asVariable**.

Dataset or CdmsFile **open(url,mode='r')** Open or create a Dataset or CdmsFile. *url* is a Uniform Resource Locator, referring to a cdunif or XML file. If the URL has the extension '.xml' or '.cdml', a Dataset is returned, otherwise a CdmsFile is returned. If the URL protocol is 'http', the file must be a '.xml' or '.cdml' file, and the mode must be 'r'. If the protocal is 'file' or is omitted, a local file or dataset is open

mode is the open mode. See Table 2.24 on page 70.

Example: Open an existing dataset:

f = cdms.open("sampleset.xml")

Example: Create a netCDF file:

f = cdms.open("newfile.nc",'w')

List **order2index (axes, orderstring)**

Find the index permutation of axes to match order. Return a list of indices

axes is a list of axis objects.

orderstring is defined as in **orderparse**.

List	**orderparse(orderstring)** Parse an order string. Returns a list of axes specifiers. *orderstring* consists of: • etters t, x, y, z meaning time, longitude, latitude, level • Numbers 0–9 representing position in axes • Dash (–) meaning insert the next available axis here. • The ellipsis ... meaning fill these positions with any remaining axes. • (name) meaning an axis whose id is name
None	**setAutoBounds(mode)** Set autobounds mode. In some circumstances CDMS can generate boundaries for 1–D axes and rectilinear grids, when the bounds are not explicitly defined. The autoBounds mode determines how this is done: If *mode* is 'grid' or 2 (the default), the **getBounds** method will automatically generate boundary information for an axis or grid if the axis is designated as a latitude or longitude axis, and the boundaries are not explicitly defined. If *mode* is 'on' or 1, the **getBounds** method will automatically generate boundary information for an axis or grid, if the boundaries are not explicitly defined. If *mode* is 'off' or 0, and no boundary data is explicitly defined, the bounds will NOT be generated; the **getBounds** method will return None for the boundaries. Note: In versions of CDMS prior to V4.0, the default mode was 'on'.
None	**setClassifyGrids(mode)** Set the grid classification mode. This affects how grid type is determined, for the purpose of generating grid boundaries. If mode is 'on' (the default), grid type is determined by a grid classification method, regardless of the value of grid.get–Type(). If mode is 'off', the value of grid.getType() determines the grid type
None	**writeScripGrid**(path, grid, gridTitle=None) Write a grid to a SCRIP grid file. *path* is a string, the path of the SCRIP file to be created. *grid* is a CDMS grid object. It may be rectangular. *gridTitle* is a string ID for the grid.

Table 2.3 Class Tags

Tag	Class
'axis'	Axis
'database'	Database
'dataset'	Dataset, CdmsFile
'grid'	RectGrid
'variable'	Variable
'xlink'	Xlink

2.4 CdmsObj

A CdmsObj is the base class for all CDMS database objects. At the application level, CdmsObj objects are never created and used directly. Rather the subclasses of CdmsObj (Dataset, Variable, Axis, etc.) are the basis of user application programming.

All objects derived from CdmsObj have a special attribute **.attributes**. This is a Python dictionary, which contains all the external (persistent) attributes associated with the object. This is in contrast to the internal, non−persistent attributes of an object, which are built−in and predefined. When a CDMS object is written to a file, the external attributes are written, but not the internal attributes.

Example: get a list of all external attributes of obj.

extatts = obj.attributes.keys()

Table 2.4 Attributes common to all CDMS objects

Type	Name	Definition
Dictionary	attributes	External attribute dictionary for this object.'

Table 2.5 Getting and setting attributes

Type	Definition
various	**value = obj.attname** Get an internal or external attribute value. If the attribute is external, it is read from the database. If the attribute is not already in the database, it is created as an external attribute. Internal attributes cannot be created, only referenced. obj.attname = value Set an internal or external attribute

value. If the attribute is external, it is written to the database.

2.5 CoordinateAxis

A CoordinateAxis is a variable that represents coordinate information. It may be contained in a file or dataset, or may be transient (memoryresident). Setting a slice of a file CoordinateAxis writes to the file, and referencing a file CoordinateAxis slice reads data from the file. Axis objects are also used to define the domain of a Variable.

CDMS defines several different types of CoordinateAxis objects. Table 2.9 on page 45 documents methods that are common to all CoordinateAxis types. Table 2.10 on page 48 specifies methods that are unique to 1D Axis objects.

Table 2.6 CoordinateAxis types

Type	Definition
CoordinateAxis	A variable that represents coordinate information. Has subtypes Axis2D and AuxAxis1D.
Axis	A one−dimensional coordinate axis whose values are strictly monotonic. Has subtypes DatasetAxis, FileAxis, and TransientAxis. May be an index axis, mapping a range of integers to the equivalent floating point value. If a latitude or longitude axis, may be associated with a RectGrid.
Axis2D	A two−dimensional coordinate axis, typically a latitude or longitude axis related to a CurvilinearGrid. Has subtypes DatasetAxis2D, FileAxis2D, and TransientAxis2D.
AuxAxis1D	A one−dimensional coordinate axis whose values need not be monotonic. Typically a latitude or longitude axis associated with a GenericGrid. Has subtypes DatasetAuxAxis1D, FileAuxAxis1D, and TransientAuxAxis1D.

An axis in a CdmsFile may be designated the unlimited axis, meaning that it can be extended in length after the initial definition. There can be at most one unlimited axis associated with a CdmsFile.

Table 2.7 CoordinateAxis Internal Attributes

Type	Name	Definition
Dictionary	attributes	External attribute dictionary.
String	id	CoordinateAxis identifer.
Dataset	parent	The dataset which contains the variable.
Tuple	shape	The length of each axis.

Table 2.8 Axis Constructors

cdms.createAxis(data, bounds=None)
Create an axis which is not associated with a dataset or file. See Table 2.2 on page 33.

Dataset.createAxis(name,ar)
Create an Axis in a Dataset. (**This function is not yet implemented.**)

CdmsFile.createAxis(name,ar,unlimited=0)
Create an Axis in a CdmsFile.
name is the string name of the Axis.
ar is a 1−D data array which defines the Axis values. It may have the value None if an *unlimited* axis is being defined.

At most one Axis in a CdmsFile may be designated as being unlimited, meaning that it may be extended in length. To define an axis as unlimited, either:

- set ar to None, and leave *unlimited* undefined, or
- set ar to the initial 1−D array, and set *unlimited* to **cdms.Unlimited**

cdms.createEqualAreaAxis(nlat)
See Table 2.2 on page 33.

cdms.createGaussianAxis(nlat)
See Table 2.2 on page 18.

cdms.createUniformLatitudeAxis(startlat, nlat, deltalat)
See Table 2.2 on page 18.

cdms.createUniformLongitudeAxis(startlon, nlon, deltalon)
See Table 2.2 on page 18.

Table 2.9 CoordinateAxis Methods

Type	Method Definition
Array	**array = axis[i:j]** Read a slice of data from the external file or dataset. Data is returned in the physical ordering defined in the dataset. See Table 2.11 on page 51 for a description of slice operators.

None **axis[i:j] = array**

Write a slice of data to the external file. **Dataset axes are read−only.**

None **assignValue(array)**

Set the entire value of the axis.

array is a Numeric array, of the same dimensionality as the axis.

Axis **clone(copyData=1)**

Return a copy of the axis, as a transient axis. If copyData is 1 (the default) the data itself is copied.

None **designateLatitude(persistent=0):**

Designate the axis to be a latitude axis.

If *persistent* is true, the external file or dataset (if any) is modified. By default, the designation is temporary.

None **designateLevel(persistent=0)**

Designate the axis to be a vertical level axis.

If *persistent* is true, the external file or dataset (if any) is modified. By default, the designation is temporary.

None **designateLongitude(persistent=0, modulo=360.0)**
Designate the axis to be a longitude axis.
modulo is the modulus value. Any given axis value x is treated as equivalent to x+modulus
If *persistent* is true, the external file or dataset (if any) is modified. By default, the designation is temporary.

None **designateTime(persistent=0, calendar = cdtime.MixedCalendar)**
Designate the axis to be a time axis.
If *persistent* is true, the external file or dataset (if any) is modified. By default, the designation is temporary.
calendar is defined as in **getCalendar()**.

Array **getBounds()**
Get the associated boundary array.

The shape of the return array depends on the type of axis:

- Axis: (n,2)
- Axis2D: (i,j,4)

- AuxAxis1D: (ncell, nvert) where nvert is the maximum number of vertices of a cell.

If the boundary array of a latitude or longitude Axis is not explicitly defined, and autoBounds mode is on, a default array is generated by calling genGenericBounds. Otherwise if auto−Bounds mode is off, the return value is None. See **setAutoBounds**.

getCalendar()

Integer

Returns the calendar associated with the (time) axis. Possible return values, as defined in the cdtime module, are:

- cdtime.GregorianCalendar: the standard Gregorian calendar
- cdtime.MixedCalendar: mixed Julian/Gregorian calendar
- cdtime.JulianCalendar: years divisible by 4 are leap years
- cdtime.NoLeapCalendar: a year is 365 days
- cdtime.Calendar360: a year is 360 days
- None: no calendar can be identified

Note: If the axis is not a time axis, the global, file−related calendar is returned.

Array **getValue()**

Get the entire axis vector.

Integer **isLatitude()**

Returns true iff the axis is a latitude axis.

Integer **isLevel()**

Returns true iff the axis is a level axis.

Integer **isLongitude()**

Returns true iff the axis is a longitude axis.

Integer **isTime()**

Returns true iff the axis is a time axis.

Integer **len(axis)**

The length of the axis if one−dimensional. If multidimensional, the length of the first dimension.

Integer **size()**

The number of elements in the axis.

String **typecode()**

The Numeric datatype identifier.

Table 2.10 Axis Methods, additional to CoordinateAxis methods

Type	Method Definition
List of component times	**asComponentTime(calendar=None)** Array version of **cdtime tocomp**. Returns a list of component times.
List of relative times	**asRelativeTime()** Array version of **cdtime torel**. Returns a list of relative times.
None	**designateCircular(modulo, persistent=0)** Designate the axis to be circular. *modulo* is the modulus value. Any given axis value x is treated as equivalent to x+modulus If *persistent* is true, the external file or dataset (if any) is modified. By default, the designation is temporary.
Integer	**isCircular()** Returns true if the axis has circular topology. An axis is defined as circular if: • axis.topology=='circular', or • axis.topology is undefined, and the axis is a longitude The default cycle for circular axes is 360.0
Integer	**isLinear()** Returns true if the axis has a linear representation.
Tuple	**mapInterval(interval)** Same as **mapIntervalExt**, but returns only the tuple (i,j), and wraparound is restricted to one cycle.
(i,j,k)	**mapIntervalExt(interval)** Map a coordinate interval to an index interval. *interval* is a tuple having one of the forms: (x,y) (x,y,indicator) (x,y,indicator,cycle)

	None or ':' where *x* and *y* are coordinates indicating the interval [x,y), and: *indicator* is a two or three−character string, where the first character is 'c' if the interval is closed on the left, 'o' if open, and the second character has the same meaning for the right−hand point. If present, the third character specifies how the interval should be intersected with the axis: • ' n' − select node values which are contained in the interval • 'b' −select axis elements for which the corresponding cell boundary intersects the interval • 'e' − same as n, but include an extra node on either side • 's' − select axis elements for which the cell boundary is a subset of the interval The default indicator is '**ccn**', that is, the interval is closed, and nodes in the interval are selected. If *cycle* is specified, the axis is treated as circular with the given cycle value. By default, if axis.isCircular() is true, the axis is treated as circular with a default modulus of 360.0. An interval of None or ':' returns the full index interval of the axis. The method returns the corresponding index interval as a 3tuple (i,j,k), where k is the integer stride, and [i.j) is the half−open index interval i<=k<j (i>=k>j if k<0), or None if the intersection is empty. For an axis which is circular (axis.topology == 'circular'), [i,j) is interpreted as follows, where N=len(axis) • if 0<=i<N and 0<=j<=N, the interval does not wrap around the axis endpoint. • otherwise the interval wraps around the axis endpoint. **See also: mapInterval, Variable.subRegion()**
TransientAxis	**subAxis(i,j,k=1)** Create an axis associated with the integer range [i:j:k]. The stride k can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

Table 2.11 Axis Slice Operators

Slice	Definition
[i]	The ith element, starting with index 0
[i:j]	The ith element through, but not including, element j
[i:]	The ith element through and including the end
[:j]	The beginning element through, but not including, element j

[:]	The entire array
[i:j:k]	Every kth element, starting at i, through but not including j
[−i]	The ith element from the end. −1 is the last element.

Example: A longitude axis has value [0.0, 2.0, ..., 358.0], of length 180. Map the coordinate interval −5.0 <= x < 5.0 to index interval(s), with wraparound. The result index interval −2<=n<3 wraps around, since −2<0, and has a stride of 1. This is equivalent to the two contiguous index intervals 2<=n<0 and 0<=n<3

```
> axis.isCircular()
1
> axis.mapIntervalExt((−5.0,5.0,'co'))
(−2,3,1)
>
```

2.6 CdmsFile

A CdmsFile is a physical file, accessible via the cdunif interface. netCDF files are accessible in read−write mode. All other formats (DRS, HDF, GrADS/GRIB, POP, QL) are accessible read−only.

As of CDMS V3, the legacy cuDataset interface is also supported by Cdms−Files. See "cu Module" on page 180.

Table 2.12 CdmsFile Internal Attributes

Type	Name	Definition
Dictionary	attributes	Global, external file attributes
Dictionary	axes	Axis objects contained in the file.
Dictionary	grids	Grids contained in the file.
String	id	File pathname.
Dictionary	variables	Variables contained in the file.

Table 2.13 CdmsFile Constructors

***fileobj* = cdms.open(path, mode)** Open the file specified by path returning a CdmsFile object. *path* is the file pathname, a string. *mode* is the open mode indicator, as listed in Table 2.24 on page 70.
***fileobj* = cdms.createDataset(path)** Create the file specified by *path*, a string.

Table 2.14 CdmsFile Methods

Type	Definition
Transient–Variable	fileobj(varname, selector) Calling a CdmsFile object as a function reads the region of data specified by the selector. The result is a transient variable, unless **raw=1** is specified. See "Selectors" on page 103. For example, the following reads data for variable 'prc', year 1980: **f = cdms.open('test.nc')** **x = f('prc', time=('1980–1','1981–1'))**
Variable, Axis, or Grid	***fileobj*['id']** Get the persistent variable, axis or grid object having the string identifier. This does not read the data for a variable. For example: **f = cdms.open('sample.nc')** **v = f['prc']** gets the persistent variable v, equivalent to **v=f.variables['prc']**. **t = f['time']** gets the axis named time, equivalent to **t=f.axes['time']**.
None	**close()** Close the file.
Axis	**copyAxis(axis, newname=None)** Copy axis values and attributes to a new axis in the file. The returned object is persistent: it can be used to write axis data to or read axis data from the file. If an axis already exists in the file, having the same name and coordinate values, it is returned. It is an error if an axis of the same name exists, but with different coordinate values. *axis* is the axis object to be copied. *newname*, if specified, is the string identifier of the new axis object. If not specified, the identifier of the input axis is used.

Grid **copyGrid(grid, newname=None)**

Copy grid values and attributes to a new grid in the file. The returned grid is persistent. If a grid already exists in the file, having the same name and axes, it is returned. An error is raised if a grid of the same name exists, having different axes.

grid is the grid object to be copied.

newname, if specified is the string identifier of the new grid object. If unspecified, the identifier of the input grid is used.

Axis **createAxis(id, ar, unlimited=0)**

Create a new Axis. This is a persistent object which can be used to read or write axis data to the file.
id is an alphanumeric string identifier, containing no blanks.

ar is the one−dimensional axis array.

Set *unlimited* to cdms.Unlimited to indicate that the axis is extensible.

RectGrid **createRectGrid(id, lat, lon, order, type="generic", mask=None)**
Create a RectGrid in the file. This is not a persistent object: the order, type, and mask are not written to the file. However, the grid may be used for regridding operations.

lat is a latitude axis in the file.

lon is a longitude axis in the file.

order is a string with value "yx" (the first grid dimension is latitude) or "xy" (the first grid dimension is longitude).

type is one of 'gaussian','uniform','equalarea',or 'generic'

If specified, *mask* is a two−dimensional, logical Numeric array (all values are zero or one) with the same shape as the grid.

Variable **createVariable(String id, String datatype,List axes, fill_value=None)**
Create a new Variable. This is a persistent object which can be used to read or write variable data to the file.
id is a String name which is unique with respect to all other objects in the file.

datatype is an MA typecode, e.g., MA.Float, MA.Int.

axes is a list of Axis and/or Grid objects.

fill_value is the missing value (optional).

Variable **createVariableCopy(var, newname=None)**

Create a new Variable, with the same name, axes, and attributes as the input variable. An error is raised if a variable of the same name exists in the file.

var is the Variable to be copied.

newname, if specified is the name of the new variable. If unspecified, the returned variable has the same name as *var*.

Note: Unlike copyAxis, the actual data is not copied to the new variable.

CurveGrid or Generic−Grid **readScripGrid**(self, whichGrid='destination', check−Grid=1)

Read a curvilinear or generic grid from a SCRIP netCDF file. The file can be a SCRIP grid file or remapping file.

If a mapping file, *whichGrid* chooses the grid to read, either **"source"** or **"destination"**.

If *checkGrid* is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. Grid cells may appear to be nonconvex if they cross a 0 / 2pi boundary. The repair consists of shifting the cell vertices to the same side modulo 360 degrees.

one **sync()**

Writes any pending changes to the file.

	write(var, attributes=None, axes=None, extbounds=None, id=None, extend=None, fill_value=None, index=None, typecode=None)
Variable	Write a variable or array to the file. The return value is the associated file variable. If the variable does not exist in the file, it is first defined and all attributes written, then the data is written. By default, the time dimension of the variable is defined as the unlimited dimension of the file. If the data is already defined, then data is extended or overwritten depending on the value of keywords *extend* and *index*, and the unlimited dimension values associated with *var*. *var* is a Variable, masked array, or Numeric array. *attributes* is the attribute dictionary for the variable. The default is var.attributes. *axes* is the list of file axes comprising the domain of the variable. The default is to copy var.getAxisList(). *extbounds* is the unlimited dimension bounds. Defaults to var.getAxis(0).getBounds() *id* is the variable name in the file. Default is var.id. *extend*=1 causes the first dimension to be unlimited: iteratively writeable. The default is None, in which case the first dimension is extensible if it is time.Set to 0 to turn off this behaviour. *fill_value* is the missing value flag. *index* is the extended dimension index to write to. The default index is determined by lookup relative to the existing extended dimension. Note: data can also be written by setting a slice of a file variable, and attributes can be written by setting an attribute of a file variable.

Table 2.15 CDMS Datatypes

CDMS Datatype	**Definition**

CdChar	character
CdDouble	double−precision floating−point
CdFloat	floating−point
CdInt	integer
CdLong	long integer
CdShort	short integer

2.7 Database

A Database is a collection of datasets and other CDMS objects. It consists of a hierarchical collection of objects, with the database being at the root, or top of the hierarchy. A database is used to:

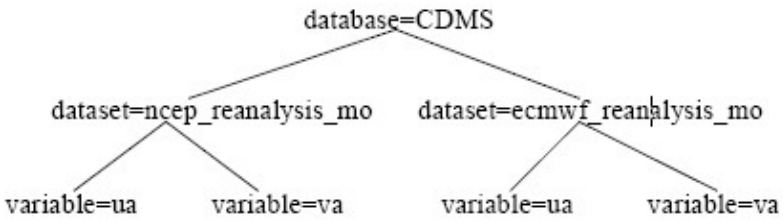
- search for metadata
- access data
- provide authentication and access control for data and metadata

The figure below illustrates several important points:

- Each object in the database has a *relative name* of the form *tag=id*. The id of an object is unique with respect to all objects contained in the parent.
- The *name* of the object consists of its relative name followed by the relative name(s) of its antecedent objects, up to and including the database name. In the figure below, one of the variables has name

"variable=ua, dataset=ncep_reanalysis_mo,database=CDMS".

- Subordinate objects are thought of as being contained in the parent. In this example, the database 'CDMS' contains two datasets, each of which contain several variables.



2.7.1 Overview

To access a database:

1. Open a connection. The **connect** method opens a database connection. **connect** takes a database URI and returns a database object:

db = cdms.connect("ldap://dbhost.llnl.gov/
database=CDMS,ou=PCMDI,o=LLNL,c=US")

2. Search the database, locating one or more datasets, variables, and/or other objects.

The database **searchFilter** method searches the database. A single database connection may be used for an arbitrary number of searches.

For example, to find all observed datasets:

result = db.searchFilter(category="observed",tag="dataset")

Searches can be restricted to a subhierarchy of the database. This example searches just the dataset 'ncep_reanalysis_mo':

result = db.searchFilter(relbase="dataset=ncep_reanalysis")

3. Refine the search results if necessary. The result of a search can be narrowed with the **searchPredicate** method.

4. Process the results. A search result consists of a sequence of entries. Each entry has a name, the name of the CDMS object, and an attribute dictionary, consisting of the attributes located by the search:

for entry in result:
print entry.name, entry.attributes

5. Access the data. The CDMS object associated with an entry is obtained from the **getObject** method:

obj = entry.getObject()

If the id of a dataset is known, the dataset can be opened directly with the **open** method:

dset = db.open("ncep_reanalysis_mo")

6. Close the database connection:

db.close()

Table 2.16 Database Internal Attributes

Type	Name	Summary
Dictionary	attributes	Database attribute dictionary
LDAP	db	(LDAP only) LDAP database object
String	netloc	Hostname, for server−based databases
String	path	path name
String	uri	Uniform Resource Identifier.

Table 2.17 Database Constructors

db = cdms.connect(uri=None, user="", password="")

Connect to the database.

uri is the Universal Resource Indentifier of the database. The form of the URI depends on the implementation of the database. For a Lightweight Directory Access Protocol (LDAP) database, the form is:

ldap://*host*[:*port*]/*dbnam*e

For example, if the database is located on host dbhost.llnl.gov, and is named 'database=CDMS,ou=PCMDI,o=LLNL,c=US', the URI is:

ldap://dbhost.llnl.gov/database=CDMS,ou=PCMDI,o=LLNL,c=US

If unspecified, the URI defaults to the value of environment variable **CDMSROOT**.

user is the user ID. If unspecified, an anonymous connection is made.

password is the user password. A password is not required for an anonymous connection.

Table 2.18 Database Methods

Type	Definition
None	close() Close a database connection.
List	listDatasets() Return a list of the dataset IDs in this database. A dataset ID can be passed to the **open** command.
Dataset	**open(dsetid, mode='r')** Open a dataset. *dsetid* is the string dataset identifier *mode* is the open mode, 'r' – read–only, 'r+' – read–write, 'w' – create.

openDataset is a synonym for **open**.

searchFilter(filter=None, tag=None, relbase=None, scope=Subtree, attnames=None, timeout=None)
Search a CDMS database.

SearchResult

filter is the string search filter. Simple filters have the form "tag = value". Simple filters can be combined using logical operators '&', '|', '!' in prefix notation. For example, the filter '(&(objectclass=variable)(id=cli))' finds all variables named "cli". A formal definition of search filters is provided in the following section.

tag restricts the search to objects with that tag ("dataset" | "variable" | "database" | "axis" | "grid").

relbase is the relative name of the base object of the search. The search is restricted to the base object and all objects below it in the hierarchy. For example, to search only dataset 'ncep_reanalysis_mo', specify: **relbase="dataset=ncep_reanalysis_mo".**

To search only variable 'ua' in 'ncep_reanalysis_mo', use: **relbase="variable=ua, dataset=ncep_reanalysis_mo"**

If no base is specified, the entire database is searched. See the *scope* argument also.

scope is the search scope (**Subtree | Onelevel | Base**). **Subtree** searches the base object and its descendants. **Onelevel** searches the base object and its immediate descendants. **Base** searches the base object alone. The default is **Subtree**.

attnames: list of attribute names. Restricts the attributes returned. If None, all attributes are returned. Attributes 'id' and 'objectclass' are always included in the list.

timeout: integer number of seconds before timeout. The default is no timeout.

2.7.2 Searching a database

The **searchFilter** method is used to search a database. The result is called a *search result*, and consists of a sequence of *result entries*.

In its simplest form, **searchFilter** takes an argument consisting of a string filter. The search returns a sequence of entries, corresponding to those objects having an attribute which matches the filter. Simple filters have the form *(tag = value)*, where *value* can contain wildcards. For example:

'(id = ncep*)'
'(project = AMIP2)'

Simple filters can be combined with the logical operators '&', '|', '!'. For example,

'(&(id = bmrc*)(project = AMIP2))'

matches all objects with id starting with bmrc, and a project attribute with value 'AMIP2'.

Formally, search filters are strings defined as follows:

filter ::= "(" filtercomp ")"

filtercomp ::= "&" filterlist | # and
"|" filterlist | # or
"!" filterlist | # not
simple

filterlist ::= filter | filter filterlist
simple ::= tag op value
op ::= "=" | # equality

"~=" | # approximate equality
"<=" | # lexicographically less than or equal to
">=" # lexicographically greater than or equal to

tag ::= string attribute name
value ::= string attribute value, may include '*' as a wild card

Attribute names are defined in the chapter on "Climate Data Markup Language (CDML)" on page 149. In addition, some special attributes are defined for convenience:

- **category** is either "experimental" or "observed"
- **parentid** is the ID of the parent dataset
- **project** is a project identifier, e.g., "AMIP2"
- **objectclass** is the list of tags associated with the object.

The set of objects searched is called the search *scope*. The top object in the hierarchy is the *base object*. By default, all objects in the database are searched, that is, the database is the base object. If the database is very large, this may result in an unnecessarily slow or inefficient search. To remedy this the search scope can be limited in several ways:

- The base object can be changed.
- The scope can be limited to the base object and one level below, or to just the base object.
- The search can be restricted to objects of a given class (dataset, variable, etc.)
- The search can be restricted to return only a subset of the object attributes
- The search can be restricted to the result of a previous search.

A search result is accessed sequentially within a for loop:

```
result = db.searchFilter('(&(category=obs*)(id=ncep*))')
for entry in result:
print entry.name
```

Search results can be narrowed using **searchPredicate**. In the following example, the result of one search is itself searched for all variables defined on a 94x192 grid:

```
>>> result = db.searchFilter('parentid=ncep*',tag=''variable'')
>>> len(result)
65
>>> result2 = result.searchPredicate(lambda x:

x.getGrid().shape==(94,192))
>>> len(result2)
3
>>> for entry in result2: print entry.name
variable=rluscs,dataset=ncep_reanalysis_mo,database=CDMS,ou=PCMDI,

o=LLNL, c=US

variable=rlds,dataset=ncep_reanalysis_mo,database=CDMS,ou=PCMDI,
o=LLNL, c=US

variable=rlus,dataset=ncep_reanalysis_mo,database=CDMS,ou=PCMDI,
o=LLNL, c=US

>>>
```

Table 2.19 SearchResult Methods

Type	Definition
ResultEntry	[i] Return the i–th search result. Results can also be returned in a for loop: for entry in db.searchResult(tag="dataset"):
Integer	len() Number of entries in the result.

SearchResult	searchPredicate(predicate, tag=None) Refine a search result, with a predicate search. predicate is a function which takes a single CDMS object and returns true (1) if the object satisfies the predicate, 0 if not. tag restricts the search to objects of the class denoted by the tag. Note: In the current implementation, searchPredicate is much less efficient than searchFilter. For best performance, use searchFilter to narrow the scope of the search, then use searchPredicate for more general searches.

A search result is a sequence of result entries. Each entry has a string name, the name of the object in the database hierarchy, and an attribute dictionary. An entry corresponds to an object found by the search, but differs from the object, in that only the attributes requested are associated with the entry. In general, there will be much more information defined for the associated CDMS object, which is retrieved with the **getObject** method.

Table 2.20 ResultEntry Attributes

Type	Name	Summary
String	name	The name of this entry in the database.
Dictionary	attributes	The attributes returned from the search. attributes[key] is a list of all string values associated with the key.

Table 2.21 ResultEntry Methods

Type	Definition
CdmsObj	**getObject**() Return the CDMS object associated with this entry. Note: For many search applications it is unnecessary to access the associated CDMS object. For best performance this function should be used only when necessary, for example, to retrieve data associated with a variable.

2.7.3 Accessing data

To access data via CDMS:

1. Locate the dataset ID. This may involve searching the metadata.
2. Open the dataset, using the open method.
3. Reference the portion of the variable to be read.

In the next example, a portion of variable 'ua' is read from dataset 'ncep_reanalysis_mo':

```
dset = db.open('ncep_reanalysis_mo')
ua = dset.variables['ua']
data = ua[0,0]
```

2.7.4 Examples of database searches

In the following examples, **db** is the database opened with

```
db = cdms.connect()
```

This defaults to the database defined in environment variable CDMSROOT.

List all variables in dataset 'ncep_reanalysis_mo':

```
for entry in db.searchFilter(filter="parentid=ncep_reanalysis_mo",
tag="variable"):
print entry.name
```

Find all axes with bounds defined:

```
for entry in db.searchFilter(filter="bounds=*",tag="axis"):
print entry.name
```

Locate all GDT datasets:

```
for entry in
db.searchFilter(filter="Conventions=GDT*",tag="dataset"):
print entry.name
```

Find all variables with missing time values, in observed datasets:

```
def missingTime(obj):
time = obj.getTime()
return time.length != time.partition_lengt

result = db.searchFilter(filter="category=observed")
for entry in result.searchPredicate(missingTime):
  print entry.name
```

Find all CMIP2 datasets having a variable with id "hfss":

```
for entry in
db.searchFilter(filter="(&(project=CMIP2)(id=hfss))",tag="var
iable"):

print entry.getObject().parent.id
```

Find all observed variables on 73x144 grids:

```
result = db.searchFilter(category='obs*')
```

```
for entry in result.searchPredicate(lambda x:
  x.getGrid().shape==(73,144),tag="variable"):
print entry.name
```

Find all observed variables with more than 1000 timepoints:

```
result = db.searchFilter(category='obs*')
for entry in result.searchPredicate(lambda x: len(x.getTime())>1000,
  tag="variable"):
print entry.name, len(entry.getObject().getTime())
```

Find the total number of each type of object in the database

```
print len(db.searchFilter(tag="database")),"database"
print len(db.searchFilter(tag="dataset")),"datasets"
print len(db.searchFilter(tag="variable")),"variables"
print len(db.searchFilter(tag="axis")),"axes"
```

2.8 Dataset

A Dataset is a virtual file. It consists of a metafile, in CDML/XML representation, and one or more data files.

As of CDMS V3, the legacy cuDataset interface is supported by Datasets. See "cu Module" on page 180.

Table 2.22 Dataset Internal Attributes

Type	Name	Summary
Dictionary	attributes	Dataset external attributes.
Dictionary	axes	Axes contained in the dataset.
String	datapath	Path of data files, relative to the parent database. If no parent, the datapath is absolute.
Dictionary	grids	Grids contained in the dataset.
String	mode	Open mode.
Database	parent	Database which contains this dataset. If the dataset is not part of a database, the value is None.
String	uri	Uniform Resource Identifier of this dataset.
Dictionary	variables	Variables contained in the dataset.
Dictionary	xlinks	External links contained in the dataset.

Table 2.23 Dataset Constructors

***datasetobj* = cdms.open(String uri, String mode='r')**

Open the dataset specified by the Universal Resource Indicator, a CDML file. Returns a Dataset object. mode is one of the indicators listed in Table 2.24 on page 70.

openDataset is a synonym for **open**.

Table 2.24 Open Modes

Mode	Definition
'r'	read−only
'r+'	read−write
'a'	read−write. Open the file if it exists, otherwise create a new file
'w'	Create a new file, read−write

Table 2.25 Dataset Methods

Type	Definition
Transient−Variable	***datasetobj*(varname, selector)** Calling a Dataset object as a function reads the region of data defined by the selector. The result is a transient variable, unless **raw=1** is specified. See "Selectors" on page 103. For example, the following reads data for variable 'prc', year 1980: **f = cdms.open('test.xml')** **x = f('prc', time=('1980−1','1981−1'))**
Variable, Axis, or Grid	***datasetobj*['id']** The square bracket operator applied to a dataset gets the persistent variable, axis or grid object having the string identifier. This does not read the data for a variable. Returns None if not found. For example: **f = cdms.open('sample.xml')** **v = f['prc']** gets the persistent variable v, equivalent to **v=f.variables['prc']**. **t = f['time']** gets the axis named 'time', equivalent to **t=f.axes['time']**

None	**close()** Close the dataset.
RectGrid	**createRectGrid(id, lat, lon, order, type="generic", mask=None)** Create a RectGrid in the dataset. This is not a persistent object: the order, type, and mask are not written to the dataset. However, the grid may be used for regridding operations. *lat* is a latitude axis in the dataset. *lon* is a longitude axis in the dataset. *order* is a string with value "yx" (the first grid dimension is latitude) or "xy" (the first grid dimension is longitude). *type* is one of 'gaussian','uniform','equalarea',or 'generic' If specified, *mask* is a two−dimensional, logical Numeric array (all values are zero or one) with the same shape as the grid.
Axis	**getAxis(id)** Get an axis object from the file or dataset. *id* is the string axis identifier.
Grid	**getGrid(id)** Get a grid object from a file or dataset. *id* is the string grid identifier.
List	**getPaths()** Get a sorted list of pathnames of datafiles which comprise the dataset. This does not include the XML metafile path, which is stored in the .uri attribute.
Variable	**getVariable(id)** Get a variable object from a file or dataset. *id* is the string variable identifier.
CurveGrid or	**readScripGrid**(self, whichGrid='destination', check−or Generic−Grid=1)

GenericGrid	Read a curvilinear or generic grid from a SCRIP dataset. The dataset can be a SCRIP grid file or remapping file. If a mapping file, *whichGrid* chooses the grid to read, either **"source"** or **"destination"**. If *checkGrid* is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. Grid cells may appear to be nonconvex if they cross a 0 / 2pi boundary. The repair consists of shifting the cell vertices to the same side modulo 360 degrees.
None	**sync()** Write any pending changes to the dataset.

2.9 MV module

The fundamental CDMS data object is the variable. A variable is comprised of:

- a masked data array, as defined in the NumPy MA module.
- a domain: an ordered list of axes and/or grids.
- an attribute dictionary.

The MV module is a work−alike replacement for the MA module, that carries along the domain and attribute information where appropriate. MV provides the same set of functions as MA. However, MV functions generate transient variables as results. Often this simplifies scripts that perform computation. MA is part of the Python Numeric package, documented at http:// numpy.sourceforge.net.

MV can be imported with the command:

import MV

The command

from MV import *

allows use of MV commands without any prefix.

Table 2.26 on page 75 lists the constructors in **MV**. All functions return a transient variable. In most cases the keywords *axes, attributes*, and *id* are available. *axes* is a list of axis objects which specifies the domain of the variable. *attributes* is a dictionary. *id* is a special attribute string that serves as the identifier of the variable, and should not contain blanks or non−printing characters. It is used when the variable is plotted or written to a file. Since the id is just an attribute, it can also be set like any attribute:

var.id = 'temperature'

For completeness MV provides access to all the MA functions. The functions not listed in the following tables are identical to the corresponding MA function: **allclose, allequal, common_fill_value, compress, create_mask, dot, e, fill_value, filled, get_print_limit, getmask, getmaskarray, identity, indices, innerproduct, isMA, isMaskedArray, is_mask, isarray, make_mask, make_mask_none, mask_or, masked, pi, put, putmask, rank, ravel, set_fill_value, set_print_limit, shape, size**. See the documentation

at http://numpy.sourceforge.net for a description of these functions.

Table 2.26 Variable Constructors in module MV

arrayrange(start, stop=None, step=1, typecode=None, axis=None, attributes=None, id=None) Just like MA.arange() except it returns a variable whose type can be specfied by the keyword argument typecode. The axis, attribute dictionary, and string identifier of the result variable may be specified. Synonym: **arange**
masked_array(a, mask=None, fill_value=None, axes=None, attributes=None, id=None) Same as MA.masked_array but creates a variable instead. If no axes are specified, the result has default axes, otherwise *axes* is a list of axis objects matching a.shape.
masked_object(data, value, copy=1, savespace=0, axes=None, attributes=None, id=None) Create variable masked where exactly data equal to value. Create the variable with the given list of axis objects, attribute dictionary, and string id.
masked_values(data, value, rtol=1e−05, atol=1e−08, copy=1, savespace=0, axes=None, attributes=None, id=None) Constructs a variable with the given list of axes and attribute dictionary, whose mask is set at those places where abs (data − value) < atol + rtol * abs (data). This is a careful way of saying that those elements of the data that have value = value (to within a tolerance) are to be treated as invalid. If data is not of a floating point type, calls masked_object instead.
ones(shape, typecode='l', savespace=0, axes=None, attributes=None, id=None) Return an array of all ones of the given length or shape.
reshape(a, newshape, axes=None, attributes=None, id=None) Copy of a with a new shape.
resize(a, new_shape, axes=None, attributes=None, id=None) Return a new array with the specified shape. The original arrays total size can be any size.
zeros(shape, typecode='l', savespace=0, axes=None, attributes=None, id=None) An array of all zeros of the given length or shape.

The following table describes the MV non−constructor functions. With the exception of **argsort**, all functions return a transient variable.

Table 2.27 MV functions

argsort(x, axis=−1, fill_value=None) Return a Numeric array of indices for sorting along a given axis.
asarray(data, typecode=None) Same as cdms.createVariable(data, typecode, copy=0). This is a short way of ensuring that something is an instance of a variable of a given type before proceeding, as in data = asarray(data) Also see the variable astype() function.
average(a, axis=0, weights=None) computes the average value of the non−masked elements of x along the selected axis. If weights is given, it must match the size and shape of x, and the value returned is: sum(a*weights)/sum(weights) In computing these sums, elements that correspond to those that are masked in x or weights are ignored.
choose(condition, t) has a result shaped like array condition. t must be a tuple of two arrays t1 and t2. Each element of the result is the corresponding element of t1 where condition is true, and the corresponding element of t2 where condition is false. The result is masked where condition is masked or where the selected element is masked.
concatenate(arrays, axis=0, axisid=None, axisattributes=None) Concatenate the arrays along the given axis. Give the extended axis the id and attributes provided − by default, those of the first array.
count(a, axis=None) Count of the non−masked elements in a, or along a certain axis.
isMaskedVariable(x) Return true if x is an instance of a variable.
masked_equal(x, value) x masked where x equals the scalar value For floating point value consider masked_values(x, value) instead.
masked_greater(x, value) x masked where x > value
masked_greater_equal(x, value) x masked where x >= value
masked_less(x, value) x masked where x < value
masked_less_equal(x, value) x masked where x <= value
masked_not_equal(x, value) x masked where x != value

masked_outside(x, v1, v2) x with mask of all values of x that are outside [v1,v2]
masked_where(condition, x, copy=1) Return x as a variable masked where *condition* is true. Also masked where x or condition masked. *condition* is a masked array having the same shape as x.
maximum(a, b=None) Compute the maximum valid values of x if y is None; with two arguments, return the element−wise larger of valid values, and mask the result where either x or y is masked.
minimum(a, b=None) Compute the minimum valid values of x if y is None; with two arguments, return the element−wise smaller of valid values, and mask the result where either x or y is masked.
outerproduct(a, b) Return a variable such that result[i, j] = a[i] * b[j]. The result will be masked where a[i] or b[j] is masked.
power(a, b) a**b
product(a, axis=0, fill_value=1) Product of elements along axis using *fill_value* for missing elements.
repeat(ar, repeats, axis=0) Return *ar* repeated *repeats* times along axis. *repeats* is a sequence of length ar.shape[axis] telling how many times to repeat each element.
set_default_fill_value(value_type, value) Set the default fill value for *value_type* to *value. value_type* is a string: 'real','complex','character','integer',or 'object'. value should be a scalar or single−element array.
sort(ar, axis=−1) Sort array ar elementwise along the specified axis. The corresponding axis in the result has dummy values.
sum(a, axis=0, fill_value=0) Sum of elements along a certain axis using *fill_value* for missing.
take(a, indices, axis=0) Return a selection of items from a. See the documentation in the Numeric manual.
transpose(ar, axes=None) Perform a reordering of the axes of array *ar* depending on the tuple of indices *axes*;thedefault is to reverse the order of the axes.
where(condition, x, y) x where *condition* is true, y otherwise.

2.10 HorizontalGrid

A HorizontalGrid represents a latitude−longitude coordinate system. In addition, it optionally describes how

lat−lon space is partitioned into cells. Specifically, a HorizontalGrid:

- consists of a latitude and longitude coordinate axis.
- may have associated boundary arrays describing the grid cell boundaries,
- may optionally have an associated logical mask.

CDMS supports several types of HorizontalGrids:

Table 2.28

GridType	Definition
RectGrid	Associated latitude an longitude are 1−D axes, with strictly monotonic values.
CurveGrid	Latitude and longitude are 2−D coordinate axes (Axis2D).
GenericGrid	Latitude and longitude are 1−D auxiliary coordinate axis (AuxAxis1D)

Table 2.29 HorizontalGrid Internal Attribute

Type	Name	Definition
Dictionary	attributes	External attribute dictionary.
String	id	The grid identifier.
Dataset or CdmsFile	parent	The dataset or file which contains the grid.
Tuple	shape	The shape of the grid, a 2−tuple.

Table 2.31 on page 82 describes the methods that apply to all types of HorizontalGrids. Table 2.32 on page 86 describes the additional methods that are unique to RectGrids.

Table 2.30 RectGrid Constructors

cdms.createRectGrid(lat, lon, order, type="generic", mask=None) Create a grid not associated with a file or dataset. See Table 2.2 on page 33.
CdmsFile.createRectGrid(id, lat, lon, order, type="generic", mask=None) Create a grid associated with a file. See Table 2.14 on page 53.
Dataset.createRectGrid(id, lat, lon, order, type="generic", mask=None) Create a grid associated with a dataset. See Table 2.25 on page 71.
cdms.createGaussianGrid(nlats, xorigin=0.0, order="yx") See Table 2.2 on page 33.
cdms.createGenericGrid(latArray, lonArray, latBounds=None, lonBounds=None, order="yx", mask=None) See Table 2.2 on page 18.

cdms.createGlobalMeanGrid(grid) See Table 2.2 on page 18.
cdms.createRectGrid(lat, lon, order, type="generic", mask=None) See Table 2.2 on page 18.
cdms.createUniformGrid(startLat, nlat, deltaLat, startLon, nlon, deltaLon, order="yx", mask=None) See Table 2.2 on page 18.
cdms.createZonalGrid(grid) See Table 2.2 on page 18.

Table 2.31 HorizontalGrid Methods

Type	Definition
Horizontal−Grid	clone() Return a transient copy of the grid.
Axis	getAxis(Integer n) Get the n−th axis. n is either 0 or 1.
Tuple	getBounds() Get the grid boundary arrays. Returns a tuple (latitudeArray, longitudeArray), where latitudeArray is a Numeric array of latitude bounds, and similarly for longitudeArray.The shape of latitudeArray and longitudeArray depend on the type of grid: * for rectangular grids with shape (nlat, nlon), the boundary arrays have shape (nlat,2) and (nlon,2). * for curvilinear grids with shape (nx, ny), the boundary arrays each have shape (nx, ny, 4). * for generic grids with shape (ncell,), the boundary arrays each have shape (ncell, nvert) where nvert is the maximum number of vertices per cell. For rectilinear grids: If no boundary arrays are explicitly defined (in the file or dataset), the result depends on the auto− Bounds mode (see cdms.setAutoBounds) and the grid classification mode (see cdms.setClassifyGrids). By default, autoBounds mode is enabled, in which case the boundary arrays are generated based on the type of grid. If disabled, the return value is (None,None). For rectilinear grids: The grid classification mode specifies how the grid type is to be determined. By default, the grid type (Gaussian, uniform, etc.) is determined by calling grid.classifyInFamily. If the mode is 'off' grid.getType is used instead

Axis	getLatitude() Get the latitude axis of this grid.
Axis	getLongitude() Get the latitude axis of this grid.
Axis	getMask() Get the mask array of this grid, if any. Returns a 2−D Numeric array, having the same shape as the grid. If the mask is not explicitly defined, the return value is None.
Axis	getMesh(self, transpose=None) Generate a mesh array for the meshfill graphics method. If transpose is defined to a tuple, say (1,0), first transpose latbounds and lonbounds according to the tuple, in this case (1,0,2).
None	setBounds(latBounds, lonBounds, persistent=0) Set the grid boundaries. latBounds is a NumPy array of shape (n,2), such that the boundaries of the kth axis value are [*latBounds*[k,0],*latBounds*[k,1]]. lonBounds is defined similarly for the longitude array. Note: By default, the boundaries are not written to the file or dataset containing the grid (if any). This allows bounds to be set on read−only files, for regridding. If the optional argument *persistent* is set to 1, the boundary array is written to the file.
None	setMask(mask, persistent=0) Set the grid mask. If persistent==1, the mask values are written to the associated file, if any. Otherwise, the mask is associated with the grid, but no I/O is generated. mask is a two−dimensional, Boolean−valued Numeric array, having the same shape as the grid.
Horizontal−Grid	**subGridRegion(latInterval, lonInterval)** Create a new grid corresponding to the coordinate region defined by latInterval, lonInterval. *latInterval* and *lonInterval* are the coordinate intervals for latitude and longitude, respectively. Each interval is a tuple having one of the forms:

	(x,y) (x,y,indicator) (x,y,indicator,cycle) None where *x* and *y* are coordinates indicating the interval [x,y), and: *indicator* is a two−character string, where the first character is 'c' if the interval is closed on the left, 'o' if open, and the second character has the same meaning for the right−hand point. (Default: 'co') If *cycle* is specified, the axis is treated as circular with the given cycle value. By default, if grid.isCircular() is true, the axis is treated as circular with a default value of 360.0. An interval of None returns the full index interval of the axis. If a mask is defined, the subgrid also has a mask corresponding to the index ranges. Note: The result grid is not associated with any file or dataset.
Transient− CurveGrid	toCurveGrid(gridid=None) Convert to a curvilinear grid. If the grid is already curvilinear, a copy of the grid object is returned. gridid is the string identifier of the resulting curvilinear grid object. If unspecified, the grid ID is copied. Note: This method does not apply to generic grids.
Transient− GenericGrid	toGenericGrid(gridid=None) Convert to a generic grid. If the grid is already generic, a copy of the grid is returned. gridid is the string identifier of the resulting curvilinear grid object. If unspecified, the grid ID is copied.

Table 2.32 RectGrid Methods, additional to HorizontalGrid Methods

String	getOrder() Get the grid ordering, either "yx" if latitude is the first axis, or "xy" if longitude is the first axis.
String	getType() Get the grid type, either "gaussian", "uniform", "equalarea", or "generic".

(Array,Array)	getWeights() Get the normalized area weight arrays, as a tuple (latWeights, lonWeights). It is assumed that the latitude and longitude axes are defined in degrees. The latitude weights are defined as: latWeights[i] = 0.5 * abs(sin(latBounds[i+1]) − sin(latBounds[i])) The longitude weights are defined as: lonWeights[i] = abs(lonBounds[i+1] − lonBounds[i])/360.0 For a global grid, the weight arrays are normalized such that the sum of the weights is 1.0 Example: Generate the 2−D weights array, such that weights[i.j] is the fractional area of grid zone [i,j]. from cdms import MV latwts, lonwts = grid.getWeights() weights = MV.outerproduct(latwts, lonwts) Also see the function area_weights in module pcmdi.weighting.
None	setType(gridtype) Set the grid type. gridtype is one of "gaussian", "uniform", "equalarea", or "generic".
RectGrid	subGrid((latStart,latStop),(lonStart,lonStop)) Create a new grid, with latitude index range [latStart : latStop] and longitude index range [lonStart : lonStop]. Either index range can also be specified as None, indicating that the entire range of the latitude or longitude is used. For example, newgrid = oldgrid.subGrid(None, (lonStart, lonStop)) creates newgrid corresponding to all latitudes, and index range [lonStart:lonStop] of oldgrid. If a mask is defined, the subgrid also has a mask corresponding to the index ranges. Note: The result grid is not associated with any file or dataset.
RectGrid	transpose() Create a new grid, with axis order reversed. The grid mask is also transposed. Note: The result grid is not associated with any file or dataset.

2.11 Variable

A Variable is a multidimensional data object, consisting of:

- a multidimensional data array, possibly masked,
- a collection of attributes
- a domain, an ordered tuple of CoordinateAxis objects.

A Variable which is contained in a Dataset or CdmsFile is called a *persistent* variable. Setting a slice of a persistent Variable writes data to the Dataset or file, and referencing a Variable slice reads data from the Dataset. Variables may also be *transient*, not associated with a Dataset or CdmsFile.

Variables support arithmetic operations. The basic Python operators are +,,*,/,**, abs, and sqrt, together with the operations defined in the **MV** module. The result of an arithmetic operation is a transient variable, that is, the axis information is transferred to the result.

The methods subRegion and subSlice return transient variables. In addition, a transient variable may be created with the **cdms.createVariable** method. The vcs and regrid module methods take advantage of the

attribute, domain, and mask information in a transient variable.

Table 2.33 Variable Internal Attributes

Type	Name	Definition
Dictionary	attributes	External attribute dictionary.
String	id	Variable identifier.
String	name_in_file	The name of the variable in the file or files which represent the dataset. If different from **id**, the variable is aliased.
Dataset or CdmsFile	parent	The dataset or file which contains the variable.
Tuple	shape	The length of each axis of the variable.

Table 2.34 Variable Constructors

Dataset.createVariable(String id, String datatype, List axes) Create a Variable in a Dataset. **This function is not yet implemented**.
CdmsFile.createVariable(String id, String datatype, List axesOr−Grids) Create a Variable in a CdmsFile. *id* is the name of the variable. *datatype* is the MA or Numeric typecode, for example, MA.Float. *axesOrGrids* is a list of Axis and/or Grid objects, on which the variable is defined. Specifying a rectilinear grid is equivalent to listing the grid latitude and longitude axes, in the order defined for the grid. Note: this argument can either be a list or a tuple. If the tuple form is used, and there is only one element, it must have a following comma, e.g.: (axisobj,).
cdms.createVariable(array, typecode=None, copy=0, savespace=0,mask=None, fill_value=None, grid=None, axes=None,attributes=None, id=None) Create a transient variable, not associated with a file or dataset. *array* is the data values: a Variable, masked array, or Numeric array. *typecode* is the MA typecode of the array. Defaults to the typecode of *array*. *copy* is an integer flag: if 1, the variable is created with a copy of the array, if 0 the variable data is shared with *array*. *savespace* is an integer flag: if set to 1, internal Numeric operations will attempt to avoid silent upcasting. *mask* is an array of integers with value 0 or 1, having the same shape as *array*. *array*

elements with a corresponding mask value of 1 are considered invalid, and are not used for subsequent Numeric operations. The default mask is obtained from array if present, otherwise is None.
fill_value is the missing value flag. The default is obtained from *array* if possible, otherwise is set to 1.0e20 for floating point variables, 0 for integer−valued variables.
grid is a rectilinear grid object.
axes is a tuple of axis objects. By default the axes are obtained from *array* if present. Otherwise for a dimension of length n, the default axis has values [0., 1., ..., double(n)].
attributes is a dictionary of attribute values. The dictionary keys must be strings. By default the dictionary is obtained from *array* if present, otherwise is empty.
id is the string identifier of the variable. By default the *id* is obtained from array if possible, otherwise is set to 'variable_n' for some integer n.

Table 2.35 Variable Methods

Type	Definition
Variable	**tvar = *var*[i:j, m:n]** Read a slice of data from the file or dataset, resulting in a transient variable. Singleton dimensions are 'squeezed' out. Data is returned in the physical ordering defined in the dataset. The forms of the slice operator are listed in Table 2.36 on page 102. ***var*[i:j, m:n] = array** Write a slice of data to the external dataset. The forms of the slice operator are listed in Table 2.21 on page 32. (Variables in CdmsFiles only)
Variable	**tvar = *var*(*selector*)** Calling a variable as a function reads the region of data defined by the *selector*. The result is a transient variable, unless **raw=1** keyword is specified. See "Selectors" on page 103.
None	**assignValue(Array ar)** Write the entire data array. Equivalent to var[:] = ar. (Variables in CdmsFiles only).
Variable	**astype(typecode)** Cast the variable to a new datatype. Typecodes are as for MV, MA, and Numeric modules.
Variable	**clone(copyData=1)** Return a copy of a transient variable. If copyData is 1 (the default) the variable data is copied as well. If copyData is 0, the result transient variable shares the original transient variables data array.

Transient Variable **crossSectionRegrid(newLevel, newLatitude, method="log", missing=None, order=None)**
Return a lat/level vertical cross−section regridded to a new set of latitudes *newLatitude* and levels *newLevel*. The variable should be a function of latitude, level, and (optionally) time.
newLevel is an axis of the result pressure levels.
newLatitude is an axis of the result latitudes.
method is optional, either "**log**" to interpolate in the log of pressure (default), or "**linear**" for linear interpolation.
missing is a missing data value. The default is var.getMissing()
order is an order string such as "tzy" or "zy". The default is var.getOrder() See also: **regrid**, **pressureRegrid**.

Axis **getAxis(n)**

Get the n−th axis.

n is an integer.

List **getAxisIds()**
Get a list of axis identifiers.

Integer **getAxisIndex(axis_spec)**
Return the index of the axis specificed by *axis_spec*. Return −1 if no match.

axis_spec is a specification as defined for **getAxisList**
getAxisList(axes=None, omit=None, order=None)
List Get an ordered list of axis objects in the domain of the variable..

If *axes* is not None, include only certain axes. Otherwise *axe*s is a list of specifications as described below. Axes are returned in the order specified unless the *order* keyword is given.

If *omit* is not None, omit those specified by an integer dimension number. Otherwise *omit* is a list of specifications as described below.

order is an optional string determining the output order.

Specifications for the axes or omit keywords are a list, each element having one of the following forms:

- an integer dimension index, starting at 0.
- a string representing an axis id or one of the strings **'time', 'latitude', 'lat', 'longitude', 'lon', 'lev'** or **'level'**.

- a function that takes an axis as an argument and returns a value. If the value returned is true, the axis matches.
- an axis object; will match if it is the same object as axis.

order can be a string containing the characters **t,x,y,z**, or **−**. If a dash ('−') is given, any elements of the result not chosen otherwise are filled in from left to right with remaining candidates.

List **getAxisListIndex(axes=None, omit=None, order=None)**
Return a list of indices of axis objects. Arguments are as for **getAxisList**.

List **getDomain()** Get the domain. Each element of the list is itself a tuple of the form

(axis,start,length,true_length) where *axis* is an axis object, *start* is the start index of the domain relative to the axis object, *length* is the length of the axis, and *true_length* is the actual number of (defined) points in the domain.
See also: **getAxisList**.

Horizontal−Grid **getGrid()** Return the associated grid, or None if the variable is not gridded.

Axis **getLatitude()**

Get the latitude axis, or None if not found.

Axis **getLevel()**

Get the vertical level axis, or None if not found.

Axis **getLongitude()**

Get the longitude axis, or None if not found.

Various **getMissing()**

Get the missing data value, or None if not found.

String **getOrder()**

Get the order string of a spatio−temporal variable. The order string specifies the physical ordering of the data. It is a string of characters with length equal to the rank of the variable, indicating the order of the variable's time, level, latitude, and/or longitude axes. Each character is one of:
't': time
'z': vertical level
'y': latitude
'x': longitude
'−': the axis is not spatio−temporal.
Example: A variable with ordering "tzyx" is 4−dimensional, where the ordering of axes is (time, level, latitude, longitude).
Note: The order string is of the form required for the *order* argument of a regridder function.

List **getPaths(*intervals)**

Get the file paths associated with the index region specified by intervals.
intervals is a list of scalars, 2−tuples representing [i,j), slices, and/or Ellipses. If no argument(s) are present, all file paths associated with the variable are returned.
Returns a list of tuples of the form (path,slicetuple), where *path* is the path of a file, and *slicetuple* is itself a tuple of slices, of the same length as the rank of the variable, representing the portion of the variable in the file corresponding to *intervals*.

Note: This function is not defined for transient variables.

Axis **getTime()**

Get the time axis, or None if not found.

Integer **len(var)**

The length of the first dimension of the variable. If the variable is zero−dimensional (scalar), a length of 0 is returned.

Note: **size()** returns the total number of elements.

Transient Variable **pressureRegrid (newLevel, method="log", missing=None, order=None)**

Return the variable regridded to a new set of pressure levels *newLevel*. The variable must be a function of latitude, longitude, pressure level, and (optionally) time.

newLevel is an axis of the result pressure levels.

method is optional, either "**log**" to interpolate in the log of pressure (default), or "**linear**" for linear interpolation.

missing is a missing data value. The default is var.getMissing()

order is an order string such as "tzyx" or "zyx". The default is var.getOrder()

See also: **regrid**, **crossSectionRegrid**.

Integer **rank()**

The number of dimensions of the variable.

regrid (togrid, missing=None, order=None, Variable **mask=None)**
Return the variable regridded to the horizontal grid *togrid*.

Transient *missing* is a Float specifying the missing data value. The default is 1.0e20.

order is a string indicating the order of dimensions of the array. It has the form returned from **variable.getOrder()**. For example, the string "tzyx" indicates that the dimension order of *array* is (time, level, latitude, longitude). If unspecified, the function assumes that the last two dimensions of *array* match the input grid.

mask is a Numeric array, of datatype Integer or Float, consisting of ones and zeros. A value of 0 or 0.0 indicates that the corresponding data value is to be ignored for purposes of regridding. If *mask* is two−dimensional of the same shape as the input grid, it overrides the mask of the input grid. If the mask has more than two dimensions, it must have the same shape as *array*. In this case, the *missing* data value is also ignored. Such an n−dimensional mask is useful if the pattern of missing data varies with level (e.g., ocean data) or time. Note: If neither *missing* or *mask* is set, the default mask is obtained from the mask of the array if any.

See also: **crossSectionRegrid**, **pressureRegrid**.

None **setAxis(n, axis)**
Set the n−th axis (0−origin index) of to a copy of *axis*.

setAxisList(axislist)

None Set all axes of the variable. *axislist* is a list of axis objects.

None **setMissing(value)**
Set the missing value.

Integer **size()**
Number of elements of the variable.

subRegion(*region, time=None, level=None, latitude=None, longitude=None, squeeze=0, raw=0)
Read a coordinate region of data, returning a transient variable. A region is a hyperrectangle in coordinate space.

Variable

region is an argument list, each item of which specifies an interval of a coordinate axis. The intervals are listed in the order of the variable axes. If trailing dimensions are omitted, all values of those dimensions are retrieved. If an axis is circular (**axis.isCircular()** is true) or cycle is specified (see below), then data will be read with wraparound in that dimension. Only one axis may be read with wraparound. A coordinate interval has one of the forms listed in Table 2.37 on page 102. Also see **axis.mapIntervalExt**.

The optional keyword arguments *time, level, latitude*, and *longitude* may also be used to specify the dimension for which the interval applies. This is particularly useful if the order of dimensions is not known in advance. An exception is raised if a keyword argument conflicts with a positional *region* argument.

The optional keyword argument *squeeze* determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 0, and such dimensions are not 'squeezed out'.

The optional keyword argument *raw* specifies whether the return object is a variable or a masked array. By default, a transient variable is returned, having the axes and attributes corresponding to2,3 the region read. If *raw*=1, an MA masked array is returned, equivalent to the transient variable without the axis and attribute information.

Variable **subSlice(*specs, time=None, level=None, latitude=None, longitude=None, squeeze=0, raw=0)**
Read a slice of data, returning a transient variable. This is a functional form of the slice operator [] with the squeeze option turned off.

specs is an argument list, each element of which specifies a slice of the corresponding dimension. There can be zero or more positional arguments, each of the form:

- (a) a single integer n, meaning slice(n, n+1)
- (b) an instance of the slice class
- (c) a tuple, which will be used as arguments to create a slice
- (d) ':', which means a slice covering that entire dimension
- (e) Ellipsis (...), which means to fill the slice list with ':' leaving only enough room at the end for the remaining positional arguments
- (f) a Python slice object, of the form slice(i,j,k)

If there are fewer slices than corresponding dimensions, all
values of the trailing dimensions are read.

The keyword arguments are defined as in **subRegion**. There must be no conflict between the positional arguments and the keywords.

In (a)−(c) and (f), negative numbers are treated as offsets from
the end of that dimension, as in normal Python indexing.

String **typecode**()
The Numeric datatype identifier.

Example: Get a region of data.

Variable ta is a function of (time, latitude, longitude). Read data corresponding to all times, latitudes −45.0 up to but not including+45.0, longitudes 0.0 through and including longitude 180.0:

data = ta.subRegion(':', (−45.0,45.0,'co'), (0.0, 180.0))

or equivalently:

data = ta.subRegion(latitude=(−45.0,45.0,'co'), longitude=(0.0, 180.0)

Read all data for March, 1980:

data = ta.subRegion(time=('1980−3','1980−4','co'))

Table 2.36 Variable Slice Operators

[i]	The ith element, zero−origin indexing.
[i:j]	The ith element through, but not including, element j
[i:]	The ith element through the end

[:j]	The beginning element through, but not including, element j
[:]	The entire array
[i:j:k]	Every kth element
[i:j, m:n]	Multidimensional slice
[i, ..., m]	(Ellipsis) All dimensions between those specified.
[−1]	Negative indices 'wrap around'. −1 is the last element.

Table 2.37 Index and Coordinate Intervals

Interval Definition		Example Interval Definition	Example
x	single point, such that axis[i]==x In general x is a scalar. If the axis is a time axis, x may also be a cdtime relative time type, component time type, or string of the form 'yyyy−mm−dd hh:mi:ss' (where trailing fields of the string may be omitted.		180.0 cdtime.reltime(48,"hours since 1980−1") '1980−1−3'
(x,y)	indices i such that x <= axis[i] <= y		(−180,180)
(x,y,'co')	x <= axis[i] < y The third item is defined as in mapInterval.		(−90,90,'cc')
(x,y,'co',cycle)	x<= axis[i] < y, with wraparound Note: It is not necesary to specify the cycle of a circular longitude axis, that is, for which axis.isCircular() is true.		(180, 180, 'co', 360.0)
slice(i,j,k)	slice object, equivalent to i:j:k in a slice operator. Refers to the indices i, i+k, i+2k, ... up to but not including index j. If i is not specified or is None it defaults to 0. If j is not specified or is None it defaults to the length of the axis. The stride k defaults to 1. k may be negative.		slice(1,10) slice(,,−1) reverses the direction of the axis.
':'	all axis values of one dimension		
Ellipsis	all values of all intermediate axes		

2.11.1 Selectors

A *selector* is a specification of a region of data to be selected from a variable. For example, the statement

x = v(time='1979−1−1', level=(1000.0,100.0))

means 'select the values of variable v for time '1979−1−1' and levels 1000.0 to 100.0 inclusive, setting x to the result.' Selectors are generally used to represent regions of space and time.

The form for using a selector is

result = v(s)

where **v** is a variable and **s** is the selector. An equivalent form is

result = f('varid', s)

where **f** is a file or dataset, and **'varid'** is the string ID of a variable.

A selector consists of a list of *selector components*. For example, the selector

time='1979−1−1', level=(1000.0,100.0)

has two components: **time='1979−1−1'**, and **level=(1000.0,100.0)**. This illustrates that selector components can be defined with keywords, using the form:

keyword=value

Note that for the keywords t**ime, level, latitude**, and **longitude**, the selector can be used with any variable. If the corresponding axis is not found, the selector component is ignored. This is very useful for writing general purpose scripts. The **required** keyword overrides this behavior. These keywords take values that are coordinate ranges or index ranges as defined in Table 2.37 on page 102.

The following keywords are available: Another form of selector components is the *positional* form, where the component order corresponds to the axis order of a variable. For example:

Table 2.38 Selector keywords

Keyword	Description	Value
axisid	Restrict the axis with ID *axisid* to a value or range of values.	See Table 2.37 on page 102
grid	Regrid the result to the grid.	Grid object
latitude	Restrict latitude values to a value or range. Short form: lat	See Table 2.37 on page 102
level	Restrict vertical levels to a value or range. Short form: lev	See Table 2.37 on page 102
longitude	Restrict longitude values to a value or range. Short form: lon	See Table 2.37 on page 102
order	Reorder the result.	Order string, e.g., "tzyx"
raw	Return a masked array (MA.array) rather than a transient variable.	0: return a transient variable (default); =1: return a masked array.
required	Require that the axis IDs be present.	List of axis identifiers.
squeeze	Remove singleton dimensions	0: leave singleton dimen

	from the result.	sions (default); 1: remove singleton dimensions.
time	Restrict time values to a value or range.	See Table 2.37 on page 102

Another form of selector components is the positional form, where the component order corresponds to the axis order of a variable. For example:

x9 = hus(('1979−1−1','1979−2−1'),1000.0)

reads data for the range **('1979−1−1','1979−2−1')** of the first axis, and coordinate value **1000.0** of the second axis. Non−keyword arguments of the form(s) listed in Table 2.37 on page 102 are treated as positional. Such selectors are more concise, but not as general or flexible as the other types described in this section.

Selectors are objects in their own right. This means that a selector can be defined and reused, independent of a particular variable. Selectors are constructed using the cdms.selectors.Selector class. The constructor takes an argument list of selector components. For example:

```
from cdms.selectors import Selector
sel = Selector(time=('1979−1−1','1979−2−1'), level=1000.)
x1 = v1(sel)
x2 = v2(sel)
```

For convenience CDMS provides several predefined selectors, which can be used directly or can be combined into more complex selectors. The selectors **time, level, latitude, longitude**, and **required** are equivalent to their keyword counterparts. For example:

```
from cdms import time, level
x = hus(time('1979−1−1','1979−2−1'), level(1000.))
```

and

```
x = hus(time=('1979−1−1','1979−2−1'), level=1000.)
```

are equivalent. Additionally, the predefined selectors **latitudeslice, longitudeslice, levelslice**, and **timeslice** take arguments (startindex, stopindex[, stride]):

```
from cdms import timeslice, levelslice
x = v(timeslice(0,2), levelslice(16,17))
```

Finally, a collection of selectors is defined in module **cdutil.region**:

```
from cdutil.region import *
NH=NorthernHemisphere=domain(latitude=(0.,90.)
SH=SouthernHemisphere=domain(latitude=(−90.,0.))
Tropics=domain(latitude=(−23.4,23.4))
NPZ=AZ=ArcticZone=domain(latitude=(66.6,90.))
SPZ=AAZ=AntarcticZone=domain(latitude=(−90.,−66.6))
```

Selectors can be combined using the & operator, or by refining them in the call:

```
from cdms.selectors import Selector
from cdms import level
sel2 = Selector(time=('1979−1−1','1979−2−1'))
sel3 = sel2 & level(1000.0)
x1 = hus(sel3)
x2 = hus(sel2, level=1000.0)
```

2.11.2 Selector examples

CDMS provides a variety of ways to select or slice data. In the following examples, variable hus is contained in file sample.nc, and is a function of (time, level, latitude, longitude). Time values are monthly starting at 1979−1−1. There are 17 levels, the last level being 1000.0. The name of the vertical level axis is 'plev'. All the examples select the first two times and the last level. The last two examples remove the singleton level dimension from the result array.

```
import cdms
f = cdms.open('sample.nc')
hus = f.variables['hus']

# Keyword selection
x = hus(time=('1979−1−1','1979−2−1'), level=1000.)
# Interval indicator (see mapIntervalExt)
x = hus(time=('1979−1−1','1979−3−1','co'), level=1000.)

# Axis ID (plev) as a keyword
x = hus(time=('1979−1−1','1979−2−1'), plev=1000.)

# Positional
x9 = hus(('1979−1−1','1979−2−1'),1000.0)

# Predefined selectors
from cdms import time, level
x = hus(time('1979−1−1','1979−2−1'), level(1000.))

from cdms import timeslice, levelslice
x = hus(timeslice(0,2), levelslice(16,17))

# Call file as a function
x = f('hus', time=('1979−1−1','1979−2−1'), level=1000.)

# Python slices
```

```
x = hus(time=slice(0,2), level=slice(16,17))

# Selector objects
from cdms.selectors import Selector
sel = Selector(time=('1979−1−1','1979−2−1'), level=1000.)
x = hus(sel)

sel2 = Selector(time=('1979−1−1','1979−2−1'))
sel3 = sel2 & level(1000.0)
x = hus(sel3)
x = hus(sel2, level=1000.0)

# Squeeze singleton dimension (level)
x = hus[0:2,16]
x = hus(time=('1979−1−1','1979−2−1'), level=1000., squeeze=1)

f.close()
```

2.12 Examples

In this example, two datasets are opened, containing surface air temperature ('tas') and upper−air temperature ('ta') respectively. Surface air temperature is a function of (time, latitude, longitude). Upper−air temperature is a function of (time, level, latitude, longitude). Time is assumed to have a relative representation in the datasets (e.g., with units 'months since basetime').

Data is extracted from both datasets for January of the first input year through December of the second input year. For each time and level, three quantities are calculated: slope, variance, and correlation. The results are written to a netCDF file. For brevity, the functions corrCoefSlope and removeSeasonalCycle are omitted.

```
1. import cdms
   import MV

   # Calculate variance, slope, and correlation of
   # surface air temperature with upper air temperature
   # by level, and save to a netCDF file. 'pathTa' is the location of
   # the file containing 'ta', 'pathTas' is the file with contains 'tas'.
   # Data is extracted from January of year1 through December of year2.
   def ccSlopeVarianceBySeasonFiltNet(pathTa,pathTas,month1,month2):

      # Open the files for ta and tas
      fta = cdms.open(pathTa)
      ftas = cdms.open(pathTas)
```

```
2.    # Get upper air temperature
      taObj = fta['ta']
      levs = taObj.getLevel()

3.    # Get the surface temperature for the closed interval
[time1,time2]
      tas = ftas('tas', time=(month1,month2,'cc'))

      # Allocate result arrays
      newaxes = taObj.getAxisList(omit='time')
      newshape = tuple([len(a) for a in newaxes])
4.    cc = MV.zeros(newshape, typecode=MV.Float, axes=newaxes,
id='correlation')
      b = MV.zeros(newshape, typecode=MV.Float, axes=newaxes,
id='slope')
      v = MV.zeros(newshape, typecode=MV.Float, axes=newaxes,
id='variance')

      # Remove seasonal cycle from surface air temperature
      tas = removeSeasonalCycle(tas)

      # For each level of air temperature, remove seasonal cycle
      # from upper air temperature, and calculate statistics
5.    for ilev in range(len(levs)):

      ta = taObj(time=(month1,month2,'cc'), \

      level=slice(ilev, ilev+1), squeeze=1)
      ta = removeSeasonalCycle(ta)
      cc[ilev], b[ilev] = corrCoefSlope(tas ,ta)
      v[ilev] = MV.sum( ta**2 )/(1.0*ta.shape[0])

      # Write slope, correlation, and variance variables
 6.   f = cdms.open('CC_B_V_ALL.nc','w')
      f.title = filtered
```

```
f.write(b)
f.write(cc)
f.write(v)
f.close()
```

7.
```
if __name__=='__main__':
pathTa = '/pcmdi/cdms/sample/ccmSample_ta.xml'
pathTas = '/pcmdi/cdms/sample/ccmSample_tas.xml'
# Process Jan80 through Dec81
ccSlopeVarianceBySeasonFiltNet(pathTa,pathTas,'80-1','81-12')
```

Notes:

1. Two modules are imported, **cdms**, and **MV**. **MV** implements arithmetic functions.
2. **taObj** is a file (persistent) variable. At this point, no data has actually been read. This happens when the file variable is sliced, or when the **subRegion** function is called. **levs** is an axis.
3. Calling the file like a function reads data for the given variable and time range. Note that **month1** and **month2** are time strings.
4. In contrast to **taObj**, the variables **cc, b**, and **v** are transient variables, not associated with a file. The assigned names are used when the variables are written.
5. Another way to read data is to call the variable as a function. The **squeeze** option removes singleton axes, in this case the level axis.
6. Write the data. Axis information is written automatically.
7. This is the main routine of the script. **pathTa** and **pathTas** pathnames. Data is processed from January 1980 through December 1981.

In the next example, the pointwise variance of a variable over time is calculated, for all times in a dataset. The name of the dataset and variable are entered, then the variance is calculated and plotted via the vcs module.

```
#!/usr/bin/env python
#
# Calculates gridpoint total variance
# from an array of interest
#

import cdms
from MV import *


# Wait for return in an interactive window

def pause():
print Hit return to continue: ,
line = sys.stdin.readline()
```

8.
```
# Calculate pointwise variance of variable over time
# Returns the variance and the number of points
# for which the data is defined, for each grid point
def calcVar(x):
```

```
    # Check that the first axis is a time axis

    firstaxis = x.getAxis(0)
    if not firstaxis.isTime():
        raise 'First axis is not time, variable:', x.id

    n = count(x,0)
    sumxx = sum(x*x)
    sumx = sum(x)
    variance = (n*sumxx -(sumx * sumx))/(n * (n-1.))

    return variance,n

    if __name__=='__main__':
    import vcs, sys


    print 'Enter dataset path [/pcmdi/cdms/obs/erbs_mo.xml]: ',
    path = string.strip(sys.stdin.readline())
    if path=='': path='/pcmdi/cdms/obs/erbs_mo.xml'

9.  # Open the dataset
    dataset = cdms.open(path)

    # Select a variable from the dataset
    print 'Variables in file:',path
    varnames = dataset.variables.keys()
    varnames.sort()
    for varname in varnames:

var = dataset.variables[varname]
if hasattr(var,'long_name'):
    long_name = var.long_name
elif hasattr(var,'title'):
    long_name = var.title
else:
    long_name = '?'

    print '%-10s: %s'%(varname,long_name)
    print 'Select a variable: ',
10. varname = string.strip(sys.stdin.readline())
    var = dataset(varname)
    dataset.close()

    # Calculate variance, count, and set attributes
    variance,n = calcVar(var)
    variance.id = 'variance_%s'%var.id
    n.id = 'count_%s'%var.id
    if hasattr(var,'units'):

  variance.units = '(%s)^2'%var.units
```

```
    # Plot variance
    w=vcs.init()
11. w.plot(variance)
    pause()
    w.clear()
    w.plot(n)
    pause()
    w.clear()
```

The result of running this script is as follows:

```
% calcVar.py
Enter dataset path [/pcmdi/cdms/sample/obs/erbs_mo.xml]:

Variables in file: /pcmdi/cdms/sample/obs/erbs_mo.xml
albt    : Albedo TOA [%]
albtcs : Albedo TOA clear sky [%]
rlcrft  : LW Cloud Radiation Forcing TOA [W/m^2]
rlut    : LW radiation TOA (OLR) [W/m^2]
rlutcs : LW radiation upward TOA clear sky [W/m^2]
rscrft : SW Cloud Radiation Forcing TOA [W/m^2]
rsdt    : SW radiation downward TOA [W/m^2]
rsut    : SW radiation upward TOA [W/m^2]
rsutcs : SW radiation upward TOA clear sky [W/m^2]
Select a variable: albt

<The variance is plotted>

Hit return to continue:

<The number of points is plotted>
```

Notes:

8. **n = count(x, 0)** returns the pointwise number of valid values, summing across axis 0, the first axis. **count** is an MV function.

9. **dataset** is a Dataset or CdmsFile object, depending on whether a .xml or .nc pathname is entered. **dataset.variables** is a dictionary mapping variable name to file variable.
10. **var** is a transient variable.
11. Plot the variance and count variables. Spatial longitude and latitude information are carried with the computations, so the continents are plotted correctly.